Eindopdracht Programmeren 5a Design Patterns

Teamleden:

- Chevan R (1072166)
- ·Amine B (1058554)
- Oumama attahiri (1064650)
- Martijn van Dijk (1086450)

Eindontwerp

In deze opdracht is Chemine Industries fabrikant van auto's en bestelbussen, die aan dealers worden verkocht. De dealers of klanten kunnen hun voertuigen samenstellen door te kiezen uit verschillende specificaties, zoals:

· Motor: Benzine, Diesel, Waterstof of Elektrisch.

· Kleur: Rood, Blauw of Zwart.

· Slot: Mechanisch, Keyless of Vingerafdruk.

· Versnellingsbak: Handgeschakeld of Automaat.

· Optionele extra's: GPS, alarmsysteem, park assist, lederen bekleding, verchroomde velgen en een spoiler.

Chemine Industries biedt dealers een eenvoudige interface om hun voertuig samen te stellen, waarna de fabriek de productie verzorgt op basis van de gemaakte keuzes. Dit proces wordt gedaan met verschillende design patterns.

1. Voertuigproductie

De productie van voertuigen wordt beheerd door een fabrieksstructuur waarin componenten modulair worden gebouwd. Dit wordt geïmplementeerd met de abstracte klasse Fabriek, die de basislogica bevat voor het samenstellen van voertuigen. De subklassen AutoFabriek en BestelbusFabriek zijn verantwoordelijk voor de productie van respectievelijk auto's en bestelbussen. De fabriek creëert voertuigen op basis van klantkeuzes:

· Motor: afhankelijk van de keuze wordt een BenzineMotor, DieselMotor, WaterstofMotor of ElektrischMotor gebruikt.

· Slot: de klant kan kiezen uit een MechanischSlot, KeylessSlot of VingerafdrukSlot.

· Kleur: er kan gekozen worden uit Rood, Blauw of Zwart.

· Versnellingsbak: keuze uit Handgeschakeld of Automaat.

De Dealer klasse roept de juiste fabriek aan op basis van het gekozen voertuigtype (auto of bestelbus).

Design Pattern: Factory Method Pattern

De Fabriek klasse en zijn subklassen (AutoFabriek en BestelbusFabriek) volgen het Factory Method Pattern. Dit patroon wordt gebruikt om objecten te creëren zonder expliciet te specificeren welke klasse wordt aangemaakt. De methoden createMotor(), createSlot(), en createVersnellingsbak() zijn voorbeelden van factory methods die specifieke componenten creëren op basis van klantkeuzes.

2. Extra's

Naast de basisconfiguratie kan een klant optionele extra's toevoegen aan het voertuig, zoals GPS, alarmsystemen, park assist, lederen bekleding, verchroomde velgen en een spoiler. Dit wordt beheerd door decoratorklassen die deze extra's dynamisch aan het voertuig toevoegen.

Design Pattern: Decorator Pattern

De manier waarop extra's worden toegevoegd, volgt het Decorator Pattern. In plaats van de voertuigklasse zelf aan te passen, worden extra's zoals GPSDecorator, AlarmsysteemDecorator en ParkAssistDecorator dynamisch "om" het voertuig gewrapt. Dit maakt het systeem flexibel en uitbreidbaar, omdat extra's kunnen worden toegevoegd zonder de onderliggende voertuiglogica te wijzigen.

3. Dealer en Fabrieksinterface

De Dealer klasse biedt een vereenvoudigde interface voor klanten om hun voertuigen samen te stellen. Op basis van de gemaakte keuzes roept de Dealer de juiste fabrieken aan (AutoFabriek of BestelbusFabriek) om het voertuig te produceren.

Design Pattern: Facade Pattern

De Dealer klasse fungeert als een vereenvoudigde interface voor het configureren van voertuigen, waardoor klanten de complexiteit van de fabriek niet hoeven te begrijpen. Dit maakt het eenvoudig voor klanten om voertuigen samen te stellen zonder in detail te weten hoe elk onderdeel werkt.

De Dealer-klasse past het Facade Pattern correct toe door de interactie met de Fabriek-klassen (de echte uitvoerende fabrieken) te beheren. Klant heeft alleen interactie met Dealer, die als tussenpersoon optreedt, en zo wordt de complexiteit voor de eindgebruiker verborgen. Dit is precies het doel van het Facade Pattern: een eenvoudige interface bieden voor een complex subsysteem.

Conclusie

Dit systeem maakt gebruik van verschillende design patterns om flexibiliteit en uitbreidbaarheid te verkrijgen. Het Factory Method Pattern zorgt ervoor dat voertuigen modulair worden opgebouwd, terwijl het Decorator Pattern dynamische aanpassingen mogelijk maakt zonder de voertuiglogica aan te passen. Het Facade Pattern zorgt ervoor dat de complexiteit voor de klant verborgen blijft, waardoor het systeem eenvoudig te gebruiken is. Deze architectuur is flexibel genoeg om eenvoudig nieuwe componenten en extra's toe te voegen, en kan zich aanpassen aan de veranderende eisen van de markt